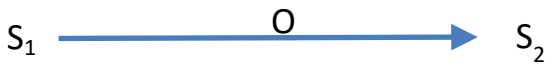
Primer on Operational Transformation

David Barrett-Lennard
15 Oct 2015

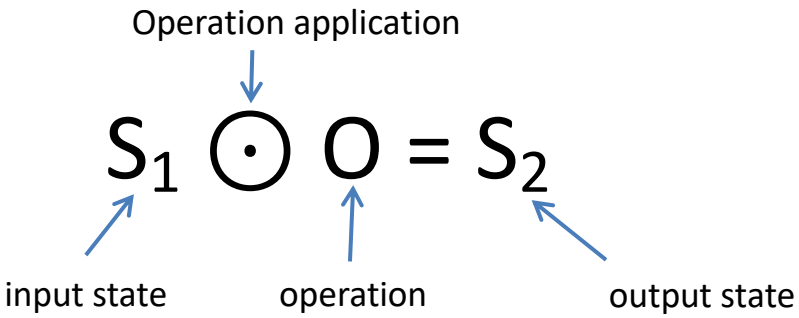
Operations

Operations represent the updates or deltas to be applied to the database. Let operation O be applied on state S_1 to get state S_2.



An example might be the deletion of a character within a string variable:

We call S_1 the ***input state*** and S_2 the ***output state*** of operation O. This state of affairs is expressed in the following equation:



Operations don't necessarily commute

Function composition is associative but not commutative.

Associative: $(f \circ g) \circ h = f \circ (g \circ h)$

Commutative: $f \circ g = g \circ f$

Similarly operations don't commute in general, meaning they can't generally be applied in different orders.

Definition: We say O_1, O_2 ***commute*** if

$$S \odot O_1 \odot O_2 = S \odot O_2 \odot O_1$$

Context equivalent and context serialised operations

Let O_1, O_2 be concurrent operations (i.e. $O_1 \,||\, O_2$).

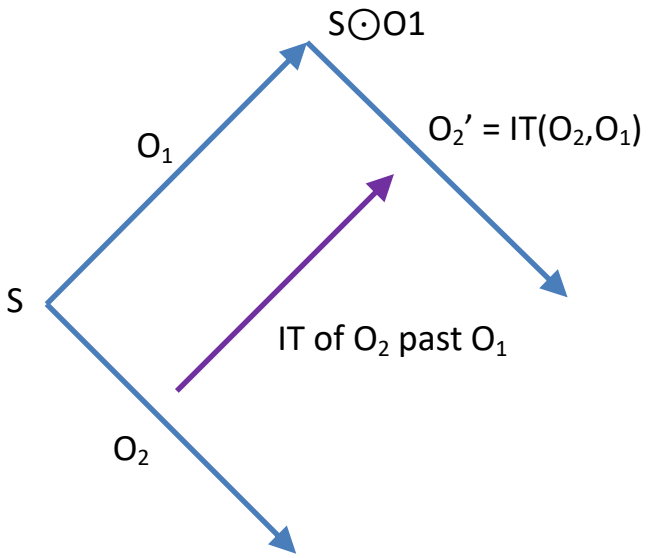
Definition: We say operations O_1 ,O_2 are ***context equivalent*** if they share the same input state.

Definition: We say operations O_1 ,O_2 are ***context serialised*** if the input state of O_2 is the output state of O_1.

Transforms

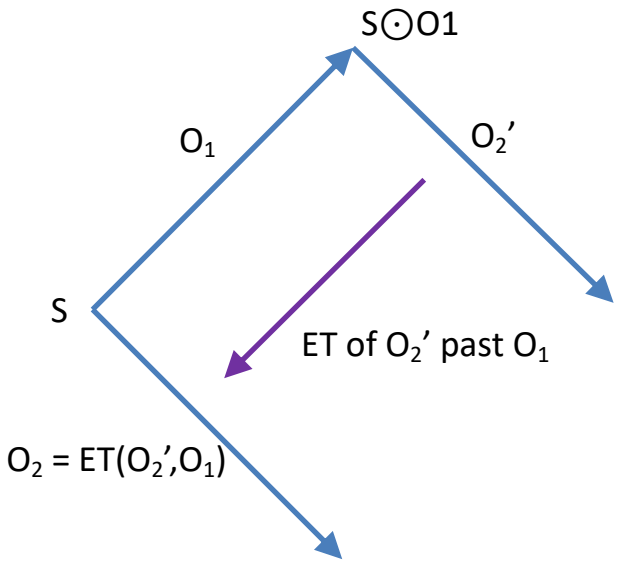
Inclusion Transformation (IT)

Let O_1 and O_2 be context equivalent operations with input state S. The following diagram depicts the ***inclusion transform*** of O_2 past O_1 to give O_2'. O_2' preserves the intention of O_2. The input state of O_2' is $S \odot O_1$.



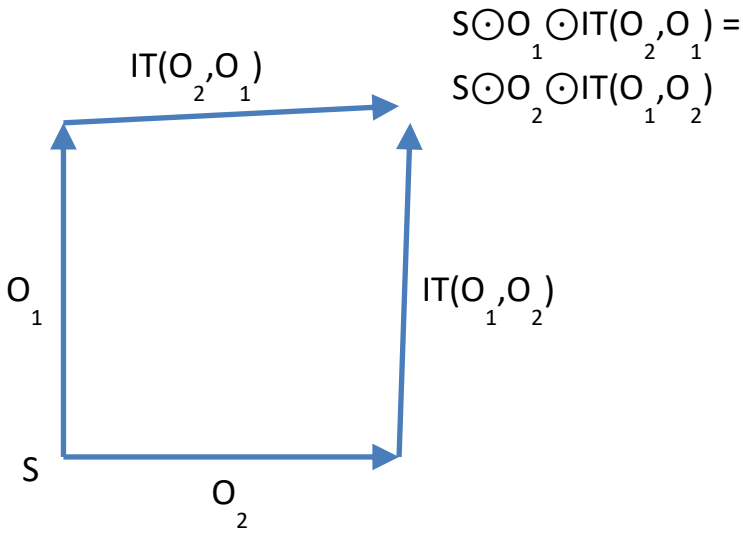
Exclusion Transformation (ET)

Let S be the input state of operation O_1 and let O_1,O_2' be context serialised. The following diagram depicts the ***exclusion transform*** of O_2' past O_1 to give O_2. O_2 preserves the intention of O_2'. The input state of O_2 is S.



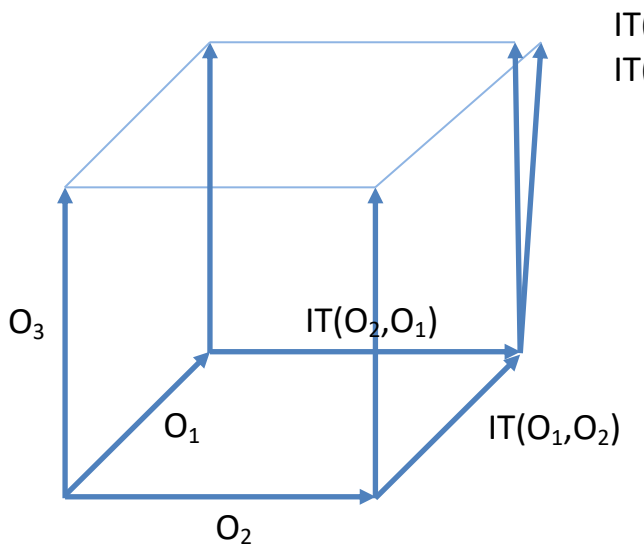
TP1

Transformation property TP1 ensures two sites converge to the same state



TP2

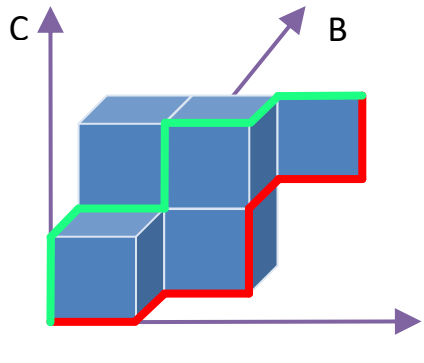
Let O_1, O_2 and O_3 be context equivalent operations. Transformation property TP2 means transforming operation O_3 either past O_1 then $IT(O_2,O_1)$ or past O_2 then $IT(O_1,O_2)$ gives the same result.



$IT(IT(O_3,O_1), IT(O_2,O_1)) =$
$IT(IT(O_3,O_2), IT(O_1,O_2))$

Sufficient conditions for convergence

TP1+TP2 $\Rightarrow$ convergence for different paths through the lattice



TP1+TP2 $\Rightarrow$ convergence

From an inductive definition of the lattice, In 1996 Ressel et al gave an inductive proof of convergence (using induction on the magnitude of the vector time).

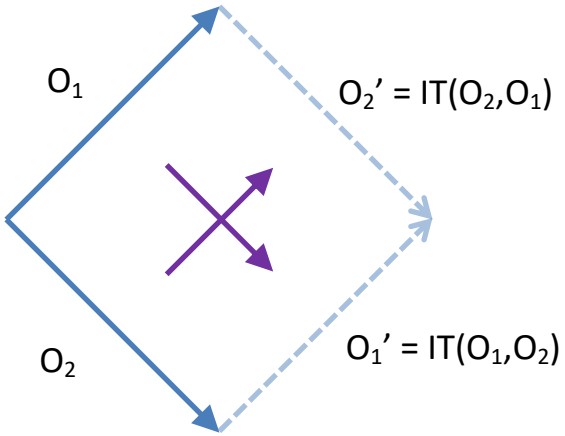
OT approaches which don't satisfy TP2

There are a number of approaches to OT which fail to satisfy condition TP2 (such as the approach used in Google Docs or Google Wave), and this imposes various limitations on the implementation:

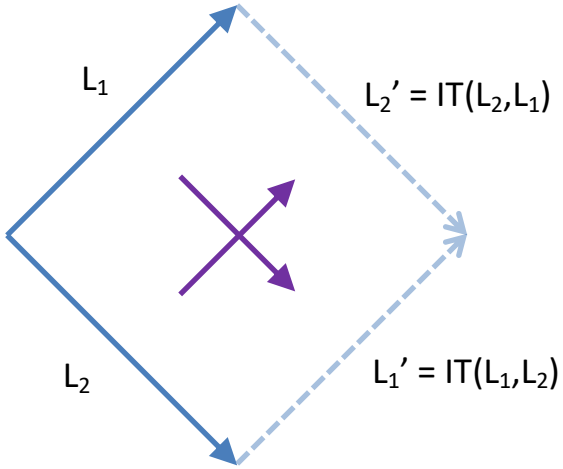
- There can be unintuitive merge results e.g. the case of three operations on text where there's an ambiguity for concurrent insertion positions on one of the faces. The solutions which fail to meet TP2 typically fail to respect a total order on insertions.
- Not robust:
  - Can't cope with server failure
  - Requires transaction durability
  - Network topology restrictions
- No arbitrary branching and merging (configuration management)

Dual IT

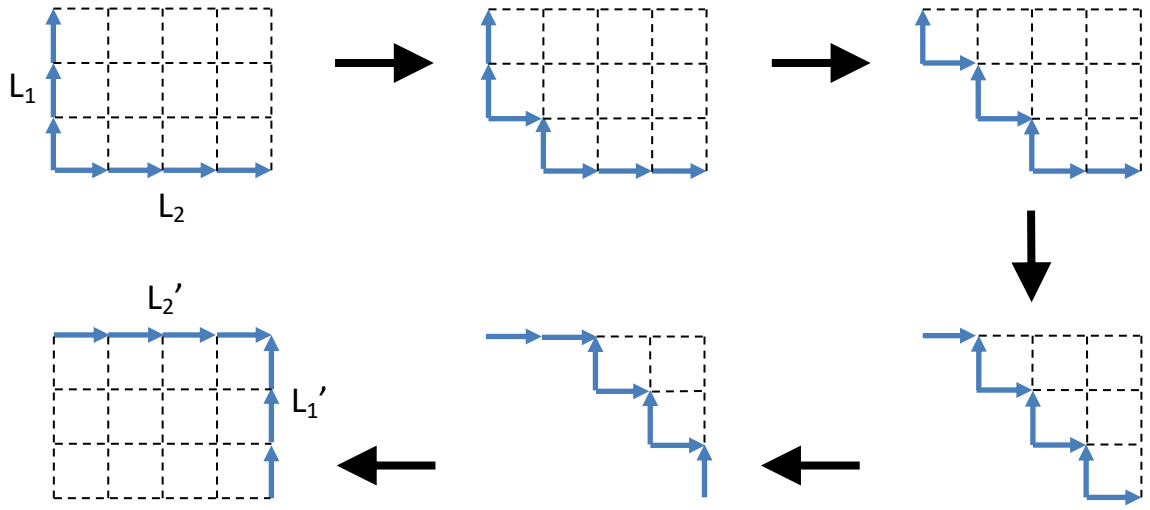
Let O_1 and O_2 be context equivalent operations. DualIT refers to the calculation of both inclusion transforms. In the diamond below the two operations on the left are transformed past each other to give the two operations on the right.



Similarly we can consider the dualIT of context equivalent lists of operations L_1, L_2 (we write $L_1 \parallel L_2$ to mean the lists are concurrent, meaning $\forall O_1 \in L_1\ \forall O_2 \in L_2\ O_1 || O_2$)

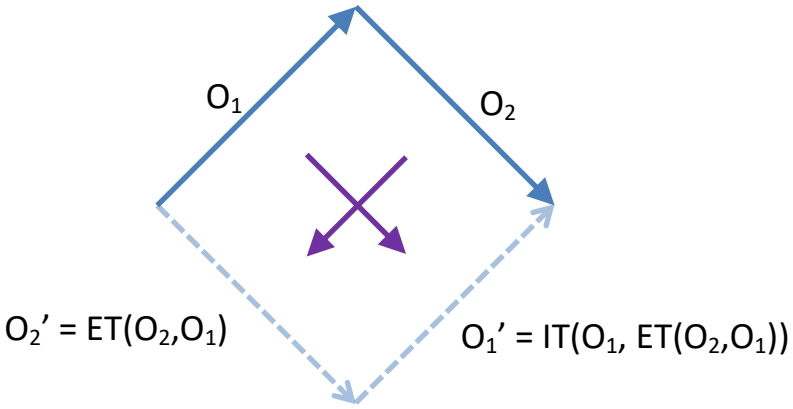


The following diagram shows how we can calculate $IT(L_2,L_1)$ and $IT(L_1,L_2)$. Unfortunately it has quadratic complexity, so it is not suitable for a practical implementation. In fact it implies the whole idea of using linear history buffers isn't suitable for implementation.

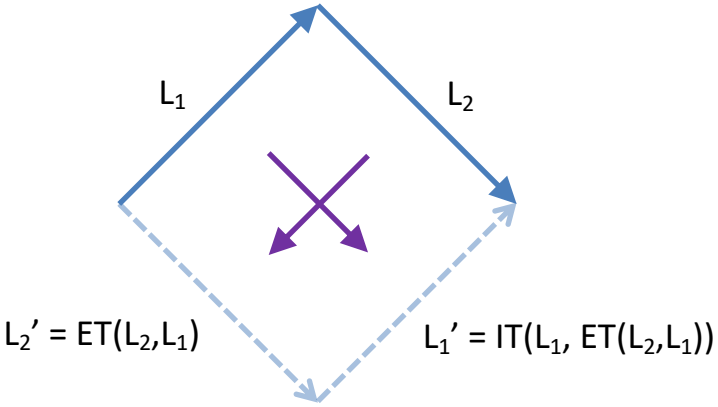


Transpose

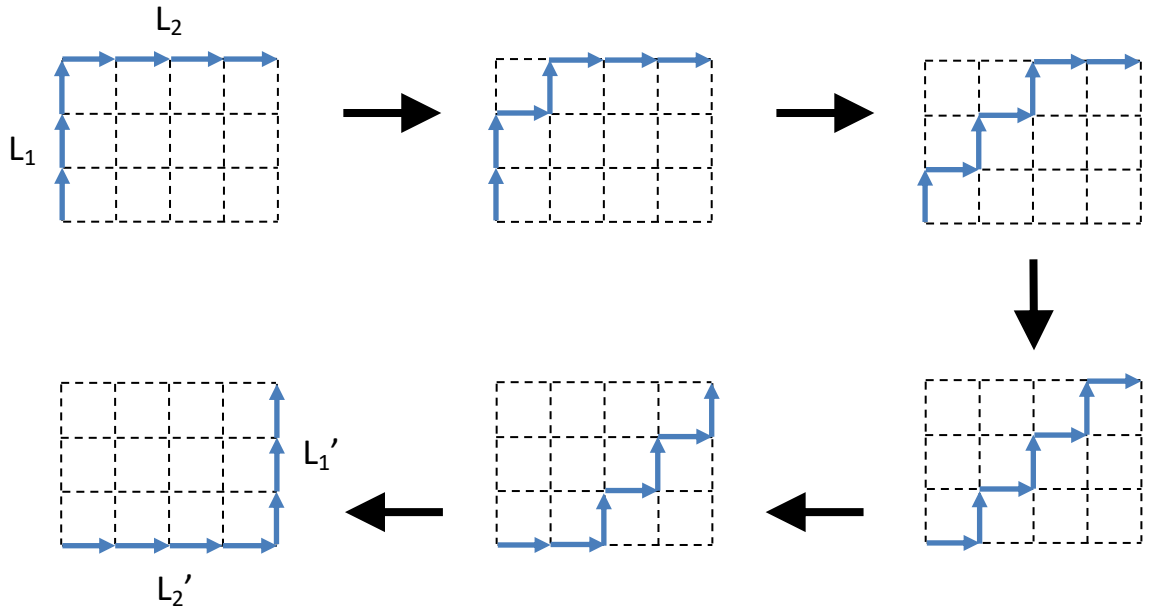
The ***transpose*** of context serialised operations O_1,O_2 is associated with applying them in the reverse order. In the following diagram the two operations on the top of the diamond are transformed to give the two operations on the bottom of the diamond.



Similarly we can consider the transpose of context serialised lists of operations L_1, L_2

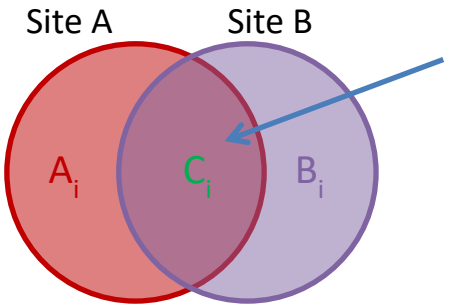


However just like DualIT the transpose of two lists is quadratic



History buffers of two sites

Consider the following Venn diagram depicting the sets of operations which have been applied so far at sites A and B

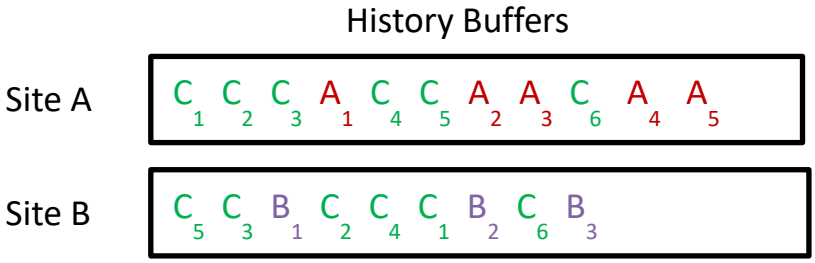


The state having the C_i preserves causality

- The Ai are the operations unique to site A
- The Bi are the operations unique to site B
- The Ci are the operations they have in common

The set of operations in common represents a state that preserves causality. Also for the union of all operations

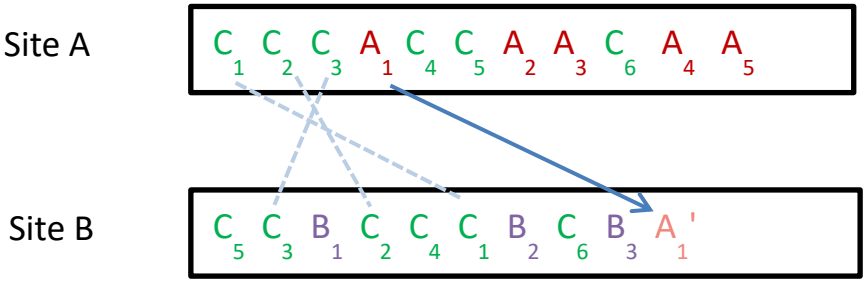
Consider that each site records the ordered list of operations it has applied so far (in the order they have been applied at that site). This list is called a ***History Buffer***. For example:



Site A has a mixture of the Ai and Ci. Site B has a mixture of the Bi and Ci.

Note that the common operations (the Ci) can be jumbled up with the Ai and Bi. Furthermore the Ci may not be in the same relative order in these two lists.

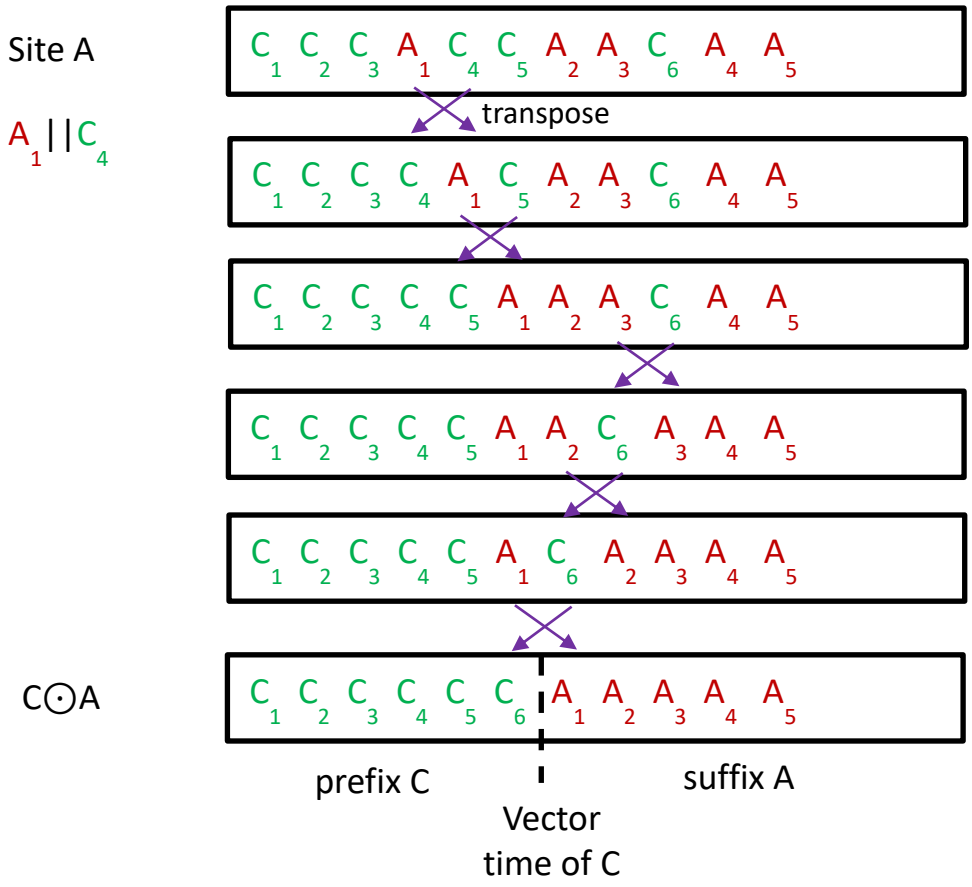
How do we synchronise the two sites? Consider for example that operation A_1 is sent to site B. The context for A_1 is the state in which $\{C_1\ C_2\ C_3\}$ have been executed. We must somehow transform A_1 to A_1' such that it can be appended to the end of site B's history buffer.



Factorise

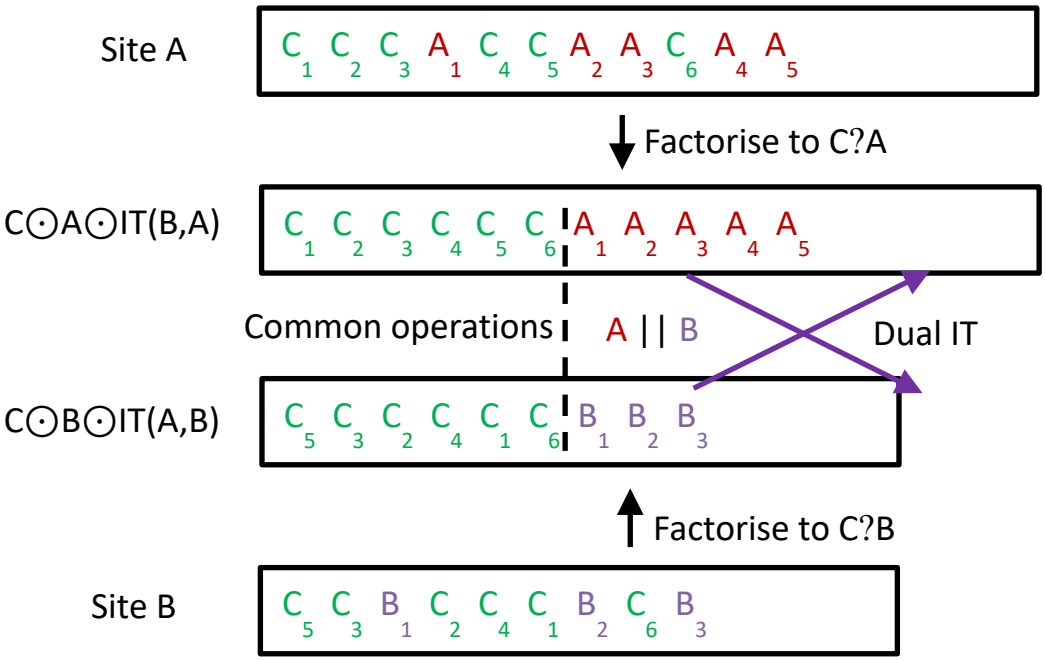
The history buffer of A has a mixture of C's and A's. We transpose adjacent pairs of Ai,Cj operations (i.e. where the Ai is to the left of the Cj) so all the C's form a prefix and the A's a suffix. This is possible because Ai || Cj. We can't have Ai→Cj because that contradicts C preserving causality, and we can't have Cj→Ai because Ai is to the left of Cj and the total order of the operations in a history buffer always respects the precedes partial order.

We call this a factorisation of the history buffer of site A with respect to a vector time V(C) to give C⊙A.



Synchronise two sites

In the following diagram the history buffers of both site A and site B are factorised so that the common operations are in a prefix. That means that even though the common operations had been applied in a different order they result in the same state. Therefore the suffixes (the A's and B's) are applied in the same context and therefore a DualIT can be performed (noting that this is possible because A || B)



But too inefficient

We have described a solution described in the literature, but it is far too inefficient to be useful in a production system, because dualIT and factorisation are quadratic algorithms.

Fortunately linear approach exist. CEDA uses algorithms that are linear in the number of operations that need to be merged. This typically allows for merging hours of works in a fraction of a second.

These linear algorithms are not described in this document.